Network Error Logging: Client-side measurement of end-to-end web service reliability

Sam Burnett[1], Lily Chen[1], Douglas A. Creager[3], Misha Efimov[1], Ilya Grigorik[1], Ben Jones[1], Harsha V. Madhyastha[1,2], Pavlos Papageorge[1], Brian Rogan[1], Charles Stahl[1], and Julia Tuttle[1]

[1]Google [2]University of Michigan [3]GitHub

nel-paper@google.com

Abstract

We present NEL (Network Error Logging), a planet-scale, client-side, network reliability measurement system. NEL is implemented in Chrome and has been proposed as a new W3C standard, letting any web site operator collect reports of clients' successful and failed requests to their sites. These reports are similar to web server logs, but include information about failed requests that never reach serving infrastructure. Reports are uploaded via redundant failover paths, reducing the likelihood of shared-fate failures of report uploads. We have designed NEL such that service providers can glean no additional information about users or their behavior compared to what services already have visibility into during normal operation. Since 2014, NEL has been invaluable in monitoring all of Google's domains, allowing us to detect and investigate instances of DNS hijacking, BGP route leaks, protocol deployment bugs, and other problems where packets might never reach our servers. This paper presents the design of NEL, case studies of real outages, and deployment lessons for other operators who choose to use NEL to monitor their traffic.

1 Introduction

Maintaining high availability is a matter of utmost importance for the operator of any popular web service. When users cannot access a web service, this may not only result in loss of revenue for the service provider but may also impact the service's reputation, causing users to shift to competing services. Therefore, it is critical that a web service operator detect and react in a timely manner when its service is inaccessible for any sizable population of users.

The primary challenge in doing so is that network traffic faces many threats as it traverses the Internet from clients to servers, most of which are outside the control of the service operator. Rogue DNS resolvers can serve hijacked results [2], ISP middleboxes can surgically alter traffic [27], bad router policy can silently drop packets [24], misconfigured BGP can take entire organizations offline [1], and more. Even though each of these issues is caused by systems that are not under the web service operator's control, the operator must bear primary responsibility for detecting and responding to them.

To address this challenge, a range of approaches have been developed over the years. For instance, server-side request logs (*e.g.*, the Apache web server's access.log and error.log files [31]) give fine-grained information about the success or failure of each incoming request. After annotating these logs with additional information, like the ISP or geographic location of the end user, operators can identify when interesting populations of end users are all affected by the same reliability issues [5, 4]. Alternative approaches rely on a dedicated monitoring infrastructure comprising a globally distributed set of vantage points. These approaches either actively probe the service to detect unreachability [6, 20] or passively monitor BGP feeds to identify routing issues [7].

Unfortunately, these existing solutions suffer from two fundamental limitations.

- First, they are typically capable of only detecting large, systemic outages. For example, with server-side monitoring, a major problem (*e.g.*, a global outage of a major service, or a regional outage affecting a large enough region) might show up as a noticeable drop in total request volume [28], but operators typically only learn of smaller problems when users manually report them.[1] These user reports are often frustratingly vague, and collecting additional information from nonexpert users is next to impossible, so investigation may take hours or even days.
- Second, and more importantly, existing approaches are incapable of precisely quantifying how many clients are affected, if any. For example, active probing from dedicated probing infrastructure can only probe from a handful of locations relative to the number of real users, and probe traffic is not always representative of what actual users experience (*e.g.*, probers may not use real web browsers and might receive different network configuration than actual end users). Without knowing how many users are affected, the operator is unable to judge whether it should prioritize the troubleshooting of a detected problem over other ongoing issues that deserve its attention.

To overcome these challenges in detecting and scoping instances of service unreachability, we need a system that (1)

[1]While sites like https://downdetector.com crowdsource such reports, historically we have often only learned of problems via social media.

passively monitors *actual* end user traffic to any target web service; (2) has visibility into reliability issues regardless of where on the end-to-end path they occur; and (3) requires little to no custom engineering work on the part of the operator.

With these goals in mind, we have designed, implemented, and deployed NEL (Network Error Logging). The key intuition behind NEL's design is that clients have ground truth about their ability to access a web service. Therefore, NEL leverages end users' browsers to collect reliability information about the service. NEL is implemented directly in the browser's network stack, collecting metadata about the outcome of requests, without requiring any custom per-site JavaScript instrumentation. The browser then uploads these reports to a redundant set of collection servers, which aggregates reports from users around the globe to detect and scope network outages.

This paper describes our contributions, based on the following three tracks:

First, we present our solutions to the various engineering and policy challenges that arise in using client-side data to detect reachability issues. When clients are unable to talk to a service, how do we still ensure successful collection of unreachability reports from these clients? What should uploaded reports contain so that they aid in diagnosing and estimating the impact of outages? How do we prevent abuse of the system, given that clients are free to upload fraudulent reports and service operators can attempt to learn about clients' reachability to other services? Our primary consideration in answering these questions has been to preserve user privacy; we ensure that NEL does not reveal more about clients than what service operators would learn during normal operation.

Second, we describe our experiences in using NEL to monitor reachability to Google's services since 2014. In that time, as we relay in Section 4, it has been instrumental in detecting and mitigating a wide variety of network outages including routing loops, BGP leaks, DNS hijacks, and protocol misconfigurations. In particular, without NEL, it would have been hard, if not impossible, to estimate the number of clients affected by each outage. Thus, NEL has proved invaluable in helping us identify which problems warrant immediate investigation due to their large impact and which ones we can afford to ignore or attend to later.

Third, after several years of experience with an initial implementation that could only monitor Google services, we describe our recent efforts to promote this capability as a new proposed W3C standard [11]. Standardizing this work has two benefits: (1) it allows all service operators to take advantage of this new collection ability, and (2) it allows operators to collect reliability data from any user agent that complies with the standard, and not just Chrome.

NEL is not a panacea for painstaking problem detection and diagnosis. It cannot report problems when clients are completely disconnected from the network or cannot reach at least one of a redundant set of collectors. It reports only coarse-grained summaries about entire requests, and is no substitute for lower-level network diagnostics like traceroutes and packet captures. (For example, it can detect when clients experience connection timeouts, but cannot tell you much more about why.) Nonetheless, NEL has proven a valuable tool for detecting and scoping network outages that are invisible to other monitoring infrastructure.

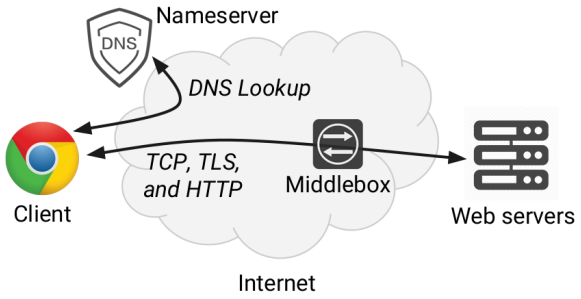

Figure 1: Steps and entities involved in enabling a client to access a web service.

2 Background and Motivation

We begin by listing several causes that may render a web service inaccessible, solutions that exist to detect service reachability problems, and the limitations of these solutions that motivated us to develop NEL.

2.1 Causes of service inaccessibility

As Figure 1 shows, a typical communication between a client and a web service offered over HTTPS involves the following steps: the client performs a DNS lookup of the service's hostname, establishes a TCP connection to the IP address it obtains, performs a TLS handshake with the server, and then sends its HTTP request.[2]

Given these steps, a client may be unable to communicate with a web service due to any of the following reasons:

- **DNS failure:** The client will be unable to execute any of the subsequent steps if its attempt to resolve the service's hostname fails. This can happen either if the nameserver that the client uses is unresponsive, or if the service provider's DNS setup is misconfigured.
- **DNS hijack:** The client could get an incorrect IP address in response to its DNS request if either the nameserver that the client uses is compromised or if the client is compromised to use a malicious nameserver.
- **IP unreachability:** When the client does get a correct IP address, the Internet may be unable to route packets from the client to that IP, either due to problems with BGP (*e.g.*, misconfiguration or convergence delays) or because of active blocking by network operators.

[2]Note that this only considers the user's communication with a "front-end" server. Modern services typically require many back-end service calls to generate the final response, which are hidden behind the interaction with the front-end server, and invisible to the client (and by extension, NEL).

- **Prefix hijack:** Alternatively, if the prefix which contains the IP address has been hijacked, the client's requests will be directed to servers not controlled by the service provider. The client will hence be (hopefully) unable to complete TLS connection setup.
- **Faulty middlebox:** When IP-level reachability between the client and the service is functional, the client's attempt to connect to the service may still fail due to a misconfigured or malicious middlebox enroute.
- **Service faulty/down:** Lastly, even if the client's DNS lookup succeeds and both IP-level and TLS-level connectivity are unhindered, the client will be unable to access a service that is down or misconfigured.

2.2 Existing solutions and limitations

To detect unreachability caused by the above-mentioned problems, a wide range of solutions have been developed over the years. This body of prior work falls broadly into four categories.

Monitoring from distributed vantage points. A popular approach is to monitor reachability from a dedicated set of distributed vantage points. Solutions that use this approach either actively probe services from devices that mimic real clients [3, 6]—probing can either be at the application level (*e.g.*, in the form of HTTP requests) or at the routing level (*e.g.*, in the form of traceroutes)—or passively monitor BGP updates (*e.g.*, to detect prefix hijacks [22, 19]), or use some combination of the two [20, 35]. Such approaches can detect problems that have wide impact. Localized problems are, however, likely to go unnoticed because a set of vantage points typically cannot match the global coverage of a service's clients. Moreover, when unreachability is detected, it is hard to estimate how many real clients are affected.

Monitoring service logs. To ensure broad coverage, service providers can monitor their service's usage either by analyzing server-side logs or by augmenting pages on their site with JavaScript that performs and uploads client-side measurements (popularly referred to as Real User Monitoring, or RUM for short [9]). With either strategy, operators must infer reachability problems from the *absence* of traffic. Requests from affected users will never arrive at the server and therefore are absent from server-side logs, whereas users unable to fetch even the HTML of a web page will not execute any JavaScript included on the page, even if cached. For any population of users, a significant drop in requests compared to what is typically expected (given historical traffic volumes) may indicate a reachability problem being experienced by these users. The challenge here is: how to distinguish between localized reachability problems and intrinsic volatility in traffic volumes? While traffic in aggregate across a large population of users is typically fairly predictable (*e.g.*, same from a particular hour in a week to that hour next week), the smaller the subset of users considered, the larger the unpredictability. As a result, drops in traffic volumes for small populations of users are not adequate evidence for service operators to take action.

Data source	Enable timely detection	Detect localized outages	Estimate # of affected clients
Distributed monitoring infrastructure	✓	×	×
Service logs	✓	×	×
Backscatter traffic	×	×	×
User reports	×	✓	×
NEL	✓	✓	✓

Table 1: Properties satisfied by different approaches for detecting service reachability problems at scale.

Monitoring backscatter traffic. Similar limitations exist with solutions that rely on backscatter traffic—traffic that clients send to unused portions of the IP address space—to detect reachability issues [12]. Here too, one has to infer (for example) censorship based on the absence of traffic. Consequently, problems can be reliably detected only when they are large in scope. Moreover, even when backscatter traffic shows an absence of traffic from a large population, those users likely cannot reach *any* IP address.

Leveraging user reports. An approach which can detect localized problems, unlike the previous categories of solutions, is to rely on complaints/reports from users. However, such solutions are typically incapable of detecting reachability problems in a timely, reproduceable, representative, and consistent manner. For example, users in some regions may be less likely to notify service operators about problems, due to language or cultural barriers.

Table 1 summarizes the limitations of these existing solutions. The overriding one, which motivated our development of NEL, is the inability to accurately estimate *how many clients are currently affected by an outage*. In our experience, a high confidence estimate of the scope of a problem is the top criterion to warrant investigation by a service operator. Lacking such data, it is hard to triage and prioritize human effort to troubleshoot the large number of issues that a service provider has to deal with at any point in time.

3 Design

This section presents the design of NEL, our browser-based mechanism for collecting information about a web service's availability, as seen by its clients. By using the client as the vantage point, the operator gains *explicit* information about the impact of reliability problems, rather then having to *estimate* the impact based on either the absence of traffic or by extrapolating from a small set of clients.

Google has twice implemented the ideas behind NEL: first as a proof of concept that could only monitor reachability to Google properties, and again as a proposed public W3C stan-

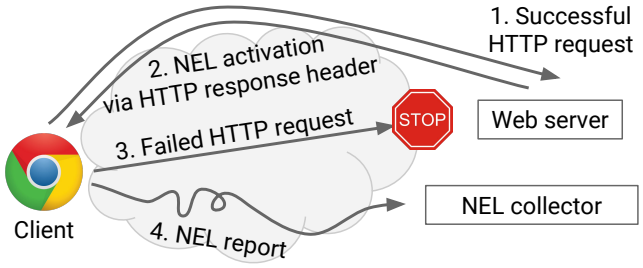

Figure 2: When a client successfully communicates with a service, collection of client-side reports is activated via a NEL policy in the service's response headers. The client reports the success and failure of its subsequent requests to that service to collectors referenced in the NEL policy.

dard [11] available to all service operators. Here, we focus on the latter due to its general availability. We also summarize important differences between the two implementations.

3.1 Approach and challenges

When stated at a high level, NEL's approach is fairly simple and intuitive: clients upload reports summarizing their ability to either successfully or unsuccessfully access target services. In aggregate, such reports enable the provider to piece together the ground truth as to how many, and which, of its clients are unable to access its service.

Realizing this approach in practice requires us to answer several questions:

- How do clients know which services to collect reliability information about?
- What should clients include in reports they upload?
- In order to reliably report that information, where should clients upload reports to?

Before describing our answers to these questions (summarized in an illustration of NEL's architecture in Figure 2), we first list the properties required of such a system, which motivate our design choices.

3.2 Security, privacy, and ethics

When designing NEL, we have to balance collecting enough information to enable quick detection of reliability issues versus satisfying the security and privacy expectations of both service owners and their end users. There are four principles in particular that we must follow:

1. We cannot collect any information about end users, their device/user agent, or their network configuration, that the server does not already have visibility into. That is, we should not collect *new information* relative to existing server logs; only existing information in a *different place*.
2. We can only collect information about requests that user agents issue when users voluntarily access services on the Web. We cannot issue requests in the background (*i.e.*, outside of normal user activity), even though this prevents us from proactively ascertaining service reachability.
3. End users can opt out of collection at any time, either globally or on a per-site basis. Support for respecting opt-outs must be implemented by NEL-compliant user agents, so that users do not need to trust service providers for opt-outs to take effect.
4. Modulo that end-user opt-out, it is *only* the site owner who gets to decide whether reports are collected about a particular site, and if so, where they are sent. Third parties (including browser vendors) must not be able to use NEL to monitor sites that they do not control.

These principles have clear ramifications on the design of the system, as we discuss below in our description of NEL's design; Table 2 provides a summary.

3.3 When do clients generate reports?

Configuration via response headers. How does a user agent know which requests to collect reports about? An operator needs a way to instruct client browsers to collect reports about requests to services they control, along with any configuration parameters about that collection.

HTTP response headers provide exactly what we need: service operators insert policy configuration headers (Report-To and NEL) into their outgoing responses; Figure 3(a) shows an example. The user agent's network stack intercepts these policy headers as part of the normal processing of the response. NEL is limited to secure connections—HTTPS connections with validated certificates—ensuring that (in the absence of a subversion of the certificate trust validation mechanism) third parties cannot inject NEL policies into the responses of servers not under their control.

If an attacker does somehow subvert connection security, they could inject NEL's HTTP headers and can obtain NEL monitoring data. For example, a malicious or compromised CDN provider could siphon NEL logs off to their own collector. But such an attacker could obtain the same data by other means (*e.g.*, by injecting additional JavaScript).

Scope of activation. We need to ensure that client-side collection of NEL reports does not allow third parties to collect information about sites they do not control. The cleanest way to do this is to follow the existing Same-Origin Policy (SOP) [8], and to have report collection be scoped to the origin (domain name, scheme, and port) of the request. That is, collection would be activated (or deactivated) for all requests to an origin; any collection policy configured for origin A would have no effect on requests to origin B, even if both origins happen to be owned and operated by the same entity.

Note that NEL does not preclude user agents from generating NEL reports even when the user is in private browsing mode. In such cases, any service's server-side logs do reflect the user's use of its service. NEL is simply collecting the same information at the client and uploading it via redundant paths to the same service provider. As we describe later in this section, the contents of any NEL report ensure

Goal	Approach	Refinements/Experience
Securely **activate client-side collection** of reachability reports	• Include NEL policy header in HTTP responses • Enforce Same-Origin Policy • Limit to HTTPS connections	• With include_subdomains option, one successful communication with an origin suffices for a user agent to start collecting (some) reports for all subdomains of that origin
Ensure report content **preserves user privacy**, yet **aids diagnosis**	• Only include information that is visible to service during normal operation • Limited set of hierarchically organized error types	• Upload reports even for successful requests to establish baseline request rate and to reduce burstiness of collection workload
Enable secure and timely **collection of NEL reports** from clients	• Causes that can render collectors unreachable should be disjoint from those that can affect the service's reachability • Client downgrades report (removing sensitive information) if service's IP address is not one from which it has received NEL policy for this origin	• Specify weight and priority per collector to balance load and to enable failover across collectors • Configurable sampling rates to reduce load on both clients and collectors • Host collectors on cloud infrastructure to deter censorship of report collection

Table 2: Summary of the approach taken in NEL to address different design decisions, along with experience-based refinements.

that the service provider can glean no additional information about its users than it can from its server-side logs.

Preventing DNS rebinding attacks. On its own, SOP is not enough to prevent a malicious actor from using NEL to learn about the reachability of origins that they do not control. Consider a *rebinding* attack, where an attacker who owns bad.example wishes to learn about the availability of good.example. They start by configuring DNS to resolve bad.example to a server that they control (using a short TTL), and getting an end user to make a request to bad.example. The server returns a NEL policy instructing the user agent to send NEL reports to a collector run by the attacker. They then update DNS to resolve bad.example to good.example's IP address(es), and cause the user to make another request to bad.example. Even though the request *looks* like it will go to the original bad.example server, it will instead be routed to good.example's server; if there are any errors with the connection, NEL reports about those errors will be sent to the attacker's collector. In this way, the attacker has been able to collect error reports about good.example, even though they do not own it. This kind of attack could also be used to port scan an internal network, by repeatedly rebinding bad.example to several different internal IP addresses.

NEL prevents such attacks by having user agents *downgrade* the quality of a report when the server IP address that a user agent contacts is not one from which it previously received the NEL policy header for this origin. In this case, instead of reporting whether the request succeeded or not (and the error type if not), the report simply states that the user agent's DNS lookup yielded a different IP address. This information is safe to report to the attacker, since it is information that they already knew; and, because it relates to what addresses bad.example resolves to, the attacker is actually the legitimate party to deliver this information to. Note that this can limit the utility of NEL's reports for domains that resolve to many IP addresses (*e.g.*, CDNs).

Subdomain reports. A consequence of activating NEL via headers in HTTP responses and enforcing SOP is that NEL cannot help an operator discover a client's inability to access their service unless the client has successfully communicated with the service at least once in the past. To minimize the impact of this constraint on service providers, a NEL policy can include the include_subdomains field, which tells the user agent to collect and upload reports for both the origin as well as all of its subdomains.

At first glance, this is a clear violation of SOP: there is no guarantee that the web sites hosted at each subdomain are owned by the same entity.[3] To maintain our privacy properties, a user agent can only use an include_subdomains policy to report *DNS errors* about requests to a subdomain, since the author of the policy has only been able to establish ownership of that portion of the domain name tree. Subdomain reports about successful requests, and about any errors that occur during or after connection establishment, are *downgraded* to reports only containing information visible in DNS. Such reports suffice for the service provider to discover unreachability due to DNS misconfiguration, *e.g.*, the provider may have forgotten to add a DNS entry for a new subdomain.

3.4 What do clients upload?

Report content. The most important part of a NEL report (Figure 3(b) shows an example) is the type, which indicates whether the underlying request succeeded or failed. If the request succeeded, the report's type will be `ok`; if it failed, it will describe what error condition caused the request to fail. The full set of predefined error types is given in the specification [11, §6]; examples include `http.error`, `dns.name_not_resolved`, `tcp.reset`, and `tcp.timed_out`. These predefined types are categorized hierarchically, so that one can find, for example, all TCP-related failures by looking for any type that starts with `tcp`.

We have found it useful to collect NEL reports even for successful requests, despite the fact that these requests also show up in server-side logs. Reporting on successful re-

[3] Consider a PaaS cloud offering like Google App Engine, where independent cloud applications are hosted at subdomains under appspot.com.

```
Report-To: {
  "endpoints":  // Try to send reports to one of these URLs
    [{"url": "https://collector1.com/upload-nel"},
     {"url": "https://collector2.com/upload-nel"}],
  "group": "nel",  // The name of this group of endpoints
  "max_age": 300  // This set of collectors expires in 5 minutes
}
NEL: {
  "failure_fraction": 1,  // Report all failed requests
  "success_fraction": 0.25,  // Report 25% of successful requests
  "include_subdomains": false,  // Don't report for subdomains
  "report_to": "nel",  // Report to a collector in this group (above)
  "max_age": 300  // This NEL policy expires in 5 minutes
}
```

(a)

```
[{
  "age": 60000,  // The report was 1 minute old when uploaded
  "type": "network-error",  // This is a NEL report
  "url": "https://example.com/about/",  // The URL the client requested
  "user_agent": "Mozilla/5.0",  // The client's User-Agent header
  "body": {
    "type": "tcp.timed_out"  // The connection timed out
    "phase": "connection",  // The request failed during handshake
    "server_ip": "203.0.113.75",  // The client tried to connect to this IP
    "sampling_fraction": 1.0,  // This report had a 100% chance of collection
    "protocol": "h2",  // The request was made using HTTP/2
    "method": "GET",
    "referrer": "https://example.com/",  // The HTTP Referer
    "elapsed_time": 45000,  // Lifetime of the request in milliseconds
  }
}]
```

(b)

Figure 3: Examples of (a) a service's use of NEL headers in its HTTPS response to activate report collection by a user agent, and (b) a report uploaded by a user agent. Comments are not included in real headers and reports.

quests lets us directly compare error ratios from successful and failed reports, without having to join the NEL logs with server-side logs. Comparing successful reports to web server logs also lets us estimate the relationship between NEL report volumes and actual request volumes.

In addition to the type, NEL reports can only contain a fixed set of additional information, as defined by the public specification. This helps ensure that implementors do not accidentally include additional information that would violate our desired privacy properties. In authoring the specification, our primary constraint when determining which fields to include is to ensure that every field in the report contains information that the server can already see during its normal processing of the request.

Given this constraint, NEL reports include basic information about the request: URL (with user credentials and fragment identifiers removed), HTTP request method (`GET`, `POST`, etc.), application and transport protocol (HTTP/1.1, HTTP/2, QUIC, etc.), User-Agent string, and referrer. The reports also include the HTTP status code of the response, if one was received, and how long the request took to complete.

Reports also contain the IP address of the server that the user agent attempted to connect to. For most requests, this is the public IP address of the service's front-end server that directly accepts incoming connections from end users. Inclusion of this IP address in reports is crucial to enable detection of DNS hijacking; though the error type in the report may indicate successful TCP connection setup, the server IP address mentioned in the report will not be one used by any of a service's front-end servers.

As mentioned above, there are several situations where a NEL report is *downgraded* for privacy reasons; for instance, when the server IP of the request is not one that the corresponding NEL policy was received from. In these cases, any privacy-sensitive fields are modified or removed from the report, to maintain the property that the report only contains information that the policy author already had access to. The NEL specification [11] contains more detail about precisely which fields are modified or removed, and when.

What do reports *not* contain? There are many details about the client that we explicitly exclude from NEL reports, even at the expense of hampering diagnosis. For example, if a user agent is using a client-configured proxy server, the IP address that the user agent attempts to connect to would be the IP address of that proxy server. Since that proxy configuration is not intended to be visible to the server, we cannot include the IP address in the report. Note that this restriction only applies to proxies configured *by the end user*. If their ISP is using a transparent proxy for all of its customers' requests, any individual user agent won't easily be able to detect this. That means that the server IP reported by the user agent will still be the actual address of the origin server, while the client IP address seen by the server and any NEL collectors will be the address of the transparent proxy.

Similarly, we cannot include the IP address of the DNS resolver that the client uses. For DNS-related network outages, this would be useful information to collect, since it would help the service owner determine whether a rogue or misconfigured DNS resolver is at fault for an outage; however, since this information is not visible to the server when processing a request, we cannot include it in a NEL report.

NEL reports also do not include details about HTTP requests that are immaterial to diagnosing reachability problems. For example, user agents exclude cookies and URL parameters from reports. A NEL report does include the full path that a request was issued for, not just the hostname to which the request was issued. We have not found much use for this information so far, but it may prove useful to an operator whose service configuration varies across different pathnames under the same origin.

Sampling rates. For high-volume sites, it is undesirable to have clients generate NEL reports about *every* attempted request, since that could double the number of requests a client must make and would require the site's collection infrastructure to be able to deal with the same full volume of request traffic as the actual site. NEL allows service operators to define a *sampling rate*, instructing user agents to only generate reports about a random subset of requests. Moreover, they can provide separate sampling rates for successful and failed requests. Typically, one will want to configure a very high sampling rate for failed requests, since those requests are more operationally relevant and more important

to collect as much information about as possible. The utility of collecting reports for successful requests is largely to estimate their total number, so lower sampling rates (*e.g.*, 1–10%) are typically sufficient. Each NEL report includes the sampling rate in effect when that particular report was generated, which allows collectors to weight each report by the inverse of its sampling rate when determining totals.

To reduce overhead, it may be tempting to adaptively vary the sampling rate over time. However, the need to increase sampling rates will arise precisely when an outage occurs. At that point, it will be infeasible for the service provider to update the NEL policy being used by affected clients.

Google uses a 5% sampling rate for successes and 100% for failures. We chose these numbers based on our experience working with NEL: (1) we find a lower sampling rate dilutes our data too much when examining small user populations (*e.g.*, when investigating outages in small ISPs), and (2) we want a relatively consistent report traffic volume, rather than massive spikes in load during major outages.

3.5 Where do clients upload reports to?

Once a user agent has generated reports about requests to an origin, those reports must somehow be sent back to the service operator's monitoring infrastructure. To do this, the service operator defines a set of *collectors*, giving an upload URL for each one (see Figure 3(a)). Since the set of collectors is defined in the NEL policy included in HTTP response headers, service operators have full control over where NEL reports about their origins are sent to.

User agents will periodically batch together reports about a particular origin, and upload those reports to one of the origin's configured collectors. The report upload is a simple `POST` request, with the JSON-serialized batched report content in the request payload.

Report uploads are subject to Cross-Origin Resource Sharing (CORS) [32] checks. If the origin of the collector is different than the origin of the requests that the NEL reports describe, the user agent will perform a CORS preflight request to verify that the collector is willing to receive NEL reports about the origin. If the CORS preflight request fails, the NEL report will be silently discarded. Reports are only uploaded over HTTPS to prevent leaking their content to passive in-network monitors.

Collector failure modes. For an operator to detect outages in a timely manner, it is crucial that clients be able to upload NEL reports even when they are unable to reach the monitored service. This requires that the collection path differ from the request path in as many ways as possible. As a consequence, not only must the collectors be hosted differently than the monitored service, but it is desirable that there be significant hosting diversity among those collectors. Examples of the ways in which collectors might differ from the monitored service include: different IP address (to learn about the service's IP being unreachable); different version of IP (if IPv4 is reachable, but not IPv6); different AS number (to account for BGP/routing issues); different transport protocol (*e.g.*, for QUIC-specific problems); and different hostname, registrar, and DNS server[4] (if the service's nameserver is unreachable). Later, in Section 4, we recount the kinds of collector diversity that have proved to be most valuable in our experience.

Given the effort necessary to ensure that collectors for a service do not share the same failure modes as the service itself, one may wonder whether the collectors could be used to improve the availability of the service, beyond collecting evidence of its (un)reachability. However, a NEL collector requires significantly fewer resources than necessary to run the monitored service. In particular, NEL collectors typically do not need to make the same latency guarantees as interactive web requests. Therefore, a service's collection infrastructure is unlikely to have the necessary capacity for an operator to serve affected users from the collectors when these users are unable to access the service normally.

Load balancing and failover. NEL enables service operators to define arbitrary load balancing and failover behavior for their collectors. Inspired by the DNS `SRV` record [17], a NEL policy can specify an optional weight and priority for each collector. When choosing which collector to upload a report to, a user agent will randomly select a collector from those with the smallest priority value, weighted by their weight values. The user agent keeps track of whether uploads to a particular collector fail too frequently; if so, that collector is marked as *pending*, and taken out of rotation. This ensures that collectors with higher priority are only attempted if *all* collectors with lower priority have failed. This mechanism gives service operators maximum flexibility in determining how to configure their collection infrastructure.

If *all* of the collectors are unreachable, the user agent will retain the reports in memory for a small amount of time (typically 15 minutes). When this happens, it often indicates a complete loss of connectivity on the part of the user. During this time, the user agent will continue attempting to deliver the reports (typically once per minute, with exponential backoffs). If the reports have still not been delivered after several attempts, they are dropped. The short interval ensures that we have visibility into temporary connectivity losses, while not requiring much storage in the user agent.

Note that a NEL user agent will also upload reports summarizing the success or failure of its attempts to upload a NEL report to a collector; after all, attempts to upload NEL reports are also HTTP requests. We refer to these as *meta reports*. Such meta reports help a service provider detect problems that clients face in contacting its collectors. To prevent infinite recursion, user agents generate meta reports only for uploads of NEL reports that are not meta reports.

[4] Note that all of these must be different for the client to use a completely different set of nameservers to resolve the collector's hostname.

Feature	Domain Reliability	NEL W3C standard
Adoption model	Opt-in	Opt-out
Activation time	Browser start	After first successful request to an origin
Activation overhead	None	HTTP response headers
Coverage	Google origins	Any HTTPS origin

Table 3: Comparison of the two implementations of NEL: the version that monitors reachability from Chrome's users to Google's services versus the W3C standard. The latter incurs additional overhead to be generic.

Censorship resistance. Although we have found NEL useful in detecting and investigating state-sponsored censorship, NEL itself is not particularly resistant to censorship. An attacker who can block access to an origin can also trivially enumerate all NEL collectors for the origin, and block access to those collectors. Service operators could try to introduce a modicum of censorship resistance by hosting NEL collectors in cloud providers, thereby tying the fate of report collection to the cloud provider as a whole. An operator could also make it harder for an adversary to identify all of its collectors by returning different subsets of collectors to different clients [34]; however, a NEL collector may be discernable via network traffic analysis [14].

Report authenticity. Since NEL reports are generated by user agents running on untrusted client devices, there is nothing preventing clients from generating and uploading fraudulent reports. A service provider can account for this challenge by only reacting to problems that affect a large enough population of unique client IP addresses (typically dozens). This measure ensures that a malicious entity can only cause a service operator to react to fake outages if they control a large number of client devices (*e.g.*, a botnet operator). However, making NEL robust to fraud in this manner comes with the risk of minimizing the impact of an outage which affects a few IP addresses shared by many client devices, *e.g.*, when many devices are behind a shared NAT.

3.6 Domain Reliability

Thus far, this section has described the public standard [11] which is usable by all services and browsers, and available in Chrome as of version 69. We also implemented these ideas in an earlier proof-of-concept called *Domain Reliability*, which could only monitor reachability from Chrome users to Google services. There are some important differences between Domain Reliability and NEL (see Table 3).

- Since Domain Reliability only monitored Google properties, and wasn't generally available to all service operators, it required users to explicitly opt in to report collection. With NEL, any user of a user agent which implements the standard is implicitly opted in to collect reports for origins which include NEL policies in their responses. The user can opt out of NEL either globally or on a per-origin basis.
- One consequence of NEL being opt-out is that its users will represent a more uniform sample of a service's userbase. Because of this, we expect to more confidently generalize results from NEL to non-NEL clients.
- With Domain Reliability, the list of origins for which clients generate reachability reports and the list of collectors to which they upload these reports was hard-coded into Chrome. Any updates to these lists were delivered as part of Chrome's regular update process, resulting in a multiple-week lead time to push any monitoring changes to our users. In contrast, with NEL, any web service gets to bootstrap the origins to monitor and the collector domains by including this information as headers in the service's HTTP responses. This allows monitoring changes to be picked up immediately, with the cost of increased overhead in client-server traffic.
- An implication of the previous point is that NEL can enable a client to upload reachability reports for a particular origin only after that client has successfully communicated with that origin at least once. Without doing so, the client would neither know that it must generate and upload reports for this origin, nor know which collectors to upload these reports to. Because its configuration was hard-coded in Chrome, Domain Reliability enabled monitoring of a client's reachability to Google's services even if that client had never successfully been able to communicate with any Google origin.
- Since Domain Reliability was only implemented in Chrome, it could not reveal reachability issues faced by users of other browsers. With NEL, a service can collect reports from any HTTP client that implements the proposed W3C standard [11].
- Because Domain Reliability's configuration was encoded in source code, we relied on the existing code review process to ensure that the configuration adhered to our desired security and privacy properties. Because this configuration was restricted (by policy) to Google properties, we did not have to downgrade Domain Reliability reports like is needed for NEL in some situations; instead, we ensured that Domain Reliability's hard-coded policy prevented reports from being sent at all in those situations.

4 Deployment Experiences

This section relays some of our experiences with the techniques described in the previous section. In each case, data alerted us to the presence of a problem and hinted at a cause based on the population of users reporting that problem (*i.e.*, the "shape" of the problem); however, these techniques are most useful in concert with other network diagnostic tools that can dig deeper into specific problems and provide "smoking gun" evidence of a particular cause.

Note that this section deals with Domain Reliability, the

```
$ traceroute X.Y.Z.33

                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- ip1.isp.net                0.0%   100    0.2   1.1   0.2  49.3   5.5
  2.|-- ip5.isp.net                0.0%   100    6.1   8.3   4.5  12.4   2.3
  3.|-- ip6.isp.net                0.0%   100    8.4   8.8   7.5  36.5   3.9
  4.|-- ip7.isp.net                0.0%   100    7.6   9.2   7.6  18.2   2.8
  5.|-- ip8.isp.net               99.0%   100 2671. 2671. 2671. 2671.   0.0
  6.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
  7.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
  8.|-- ip9.isp.net               99.0%   100 7314. 7314. 7314. 7314.   0.0
  9.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
 10.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
 11.|-- ip10.isp.net              99.0%   100 5179. 5179. 5179. 5179.   0.0
 12.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
 13.|-- ip11.isp.net              99.0%   100 2722. 2722. 2722. 2722.   0.0
 14.|-- ???                      100.0   100    0.0   0.0   0.0   0.0   0.0
```

(a)

```
$ traceroute X.Y.Z.32

                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- ip1.isp.net                0.0%    10    0.2   0.2   0.2   0.3   0.0
  2.|-- ip2.isp.net                0.0%    10    6.9   8.4   4.8  11.8   2.4
  3.|-- ip3.isp.net                0.0%    10    7.7   8.3   7.5  10.7   1.1
  4.|-- ip4.isp.net                0.0%    10   33.5  10.7   7.5  33.5   8.1
  5.|-- ip1.google.com              0.0%    10   49.7  49.2  43.3  55.2   4.4
  6.|-- ip2.google.com              0.0%    10   49.0  48.5  44.3  51.6   2.6
  7.|-- ip3.google.com              0.0%    10   47.6  47.8  44.3  53.2   2.9
  8.|-- X.Y.Z.32                    0.0%    10   48.9  49.8  45.9  58.7   4.6
```

(b)

Figure 4: (a) Traceroute to the affected IP address appears to show a routing loop in the last-mile ISP, and (b) traceroute from the same vantage point to an adjacent IP address exits the ISP within a few hops. Hostnames and IP addresses are anonymized.

initial prototype of these concepts that we developed to monitor traffic only to Google's services. We have monitored Google's services with it since Chrome 38 in 2014 and are currently migrating our monitoring to use NEL. Although Domain Reliability and NEL are not qualitatively different, Section 3.6 explains how one might expect our experiences to differ once we fully migrate to NEL.

4.1 Unreachability of a single IP address

In December of 2018, Chrome clients started reporting failures of TCP connections and QUIC sessions made to a single Google IP address. The problem affected all requests to that IP from all users in every ISP in one country for the following two weeks.

Further manual investigation revealed that traceroutes from an affected host to that IP were failing inside a transit ISP which dominates wired connectivity within that country. We also occasionally noticed IPs belonging to this transit provider in high-TTL hops of the traceroute, suggesting that packets were stuck in a routing loop in that ISP's network (see Figure 4). The problem was eventually resolved after we contacted the ISP, although we never learned how they presumably fixed it.

NEL was useful in this case in several ways.

- It let us quickly detect a problem we may have never noticed otherwise. The impacted IP served content that is visible to users but is not critical (*e.g.*, thumbnail images), which may be why we received no reports about this problem on social media, and why there was no visibility on crowdsourced sites like downdetector.com.
- NEL was additionally helpful in confirming that the problem had gone away, particularly since the ISP never notified us that they fixed it.
- Diversity of our NEL collectors was trivially useful in this case; a collector hosted on *any* IP other than the one impacted could have successfully received report uploads.

Although other network monitoring tools could have caught this problem (*e.g.*, active probing of all serving IPs), it would have been difficult to assess the scale of users impacted or even to achieve dense enough probe coverage to know how many ISPs were affected. It would have been even more difficult to feel confident that the problem had been completely resolved.

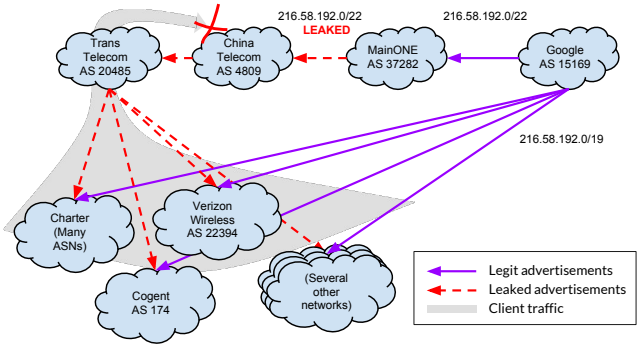

Figure 5: AS 37252 leaked prefixes to its peers, which ultimately misrouted traffic in several downstream ASNs. NEL quantified the impact of the leak on users in those downstream networks.

However, NEL alone was not enough to diagnose this issue because it gave no information about the location of the problem other than the set of users affected. We needed other tools, like traceroute, to identify which network links were impacted and that most, if not all, of the Internet traffic in that country transits through one ISP.

4.2 BGP leakage

On November 12, 2018, AS 37282 leaked its routing table to its upstream providers. As shown in Figure 5, this leak rerouted traffic destined for some Google prefixes, causing packet loss for many of our users. The network operations community noticed this incident and the media widely reported on it.[5]

NEL saw this incident as an increase in connection timeouts for the leaked prefixes. Although many other monitoring tools had clear visibility of the leaked BGP routes [7], NEL directly told us the incident's impact on user traffic. In some networks, NEL reported that nearly all requests to these prefixes timed out. Moreover, diversity of our NEL collectors was useful in this case, because we had collectors running in IP prefixes not affected by the leakage.

[5] https://www.manrs.org/2018/11/route-leak-causes-major-google-outage/

On the other hand, NEL said nothing about the cause of the outage other than perhaps that the problem was localized to a specific set of prefixes. It would probably be most valuable to overlay NEL data on existing BGP leak detection tools, to distinguish relatively benign incidents of BGP leakage (*i.e.*, that do not impact much traffic) from major events like this one.

Note that BGP leaks that have no impact on the users of Google's services are likely since Google advertises and uses different IP prefixes to serve its users in different parts of the world. Hence, when an ISP erroneously advertises one of our prefixes, this has no impact on users if this ISP operates in a region far from where we use the affected prefix. NEL helps us avoid troubleshooting such inconsequential problems, whose impact would otherwise have been hard to determine only based on analysis of BGP data.

4.3 Malware-induced DNS hijacking

In February of 2018, NEL reported that users in a large ISP were resolving several of our domains to non-authoritative IP addresses belonging to a third-party cloud hosting provider. Clients could still complete requests to these domains (*i.e.*, most reports we received had type `ok`), suggesting the presence of a layer 3 proxy server. Although requests were successful, this was still concerning because a proxy could reduce performance and reliability. We could not find reports of the problem online or reproduce the issue ourselves from our CDN infrastructure in the ISP.

We later discovered a rogue DNS resolver associated with malware that was responding to DNS requests in much the same way. We theorize that either (1) an ISP DNS server had been compromised to resolve requests using the rogue resolver, or (2) many machines and/or home routers in that ISP had been similarly compromised.

This case highlights a key advantage of NEL over traditional active probing: NEL monitors actual user network conditions and configurations, which can differ from those of dedicated monitoring infrastructure. It can detect massive problems that are simply invisible to other forms of monitoring. However, this case also highlights that NEL is not necessarily very helpful in debugging problems; in this case, since NEL does not report which DNS resolver clients use, it did not help us make progress on the problem.

Note that collector diversity was unnecessary in this case because requests to these hijacked domains still worked. Nonetheless, NEL was helpful because it alerted us to the presence of a proxy server associated with malware.

4.4 Protocol-specific problems

On March, 17, 2017, NEL observed that users were having trouble connecting to Google services in two of our datacenters in the United States and Europe. On closer inspection, only clients using QUIC [23] were affected. This was corroborated by reports from users.[6]

Our operations team traced the problem back to a bad server configuration change, and mitigated it soon after. The problem caused machines in the affected datacenters to drop all traffic on established QUIC sessions. Although QUIC clients transparently fall back to TCP when QUIC cannot establish a connection, that did not help in this case because we only dropped packets *after* a connection was established.

This situation illustrates the value of black box traffic monitoring; if operations teams only monitor specific protocol-level metrics (*e.g.*, number of connections established), then there is a chance that those metrics do not tell the whole story. NEL lets operations teams know whether users' connections are healthy end-to-end.

NEL collector diversity was useful in this case because the problem was localized to a few datacenters; clients could successfully upload NEL reports to other locations.

4.5 Monitoring of NEL report uploads

In addition to discovering network outages, we have also leveraged NEL's collection infrastructure to monitor previously unmonitored infrastructure. Other operators of NEL collectors may do the same.

Domain Reliability monitors only Google's first-party services, but not customer-owned origins hosted on Google's cloud infrastructure. This is currently a blind spot in our monitoring. Moreover, NEL will not allow us to monitor customer-owned origins directly, since NEL's privacy design gives the *customer*, and not their cloud provider, control over whether reports are collected and where they are sent.

To help ameliorate this limitation, in addition to our existing diverse set of NEL collectors, we run another set of collectors hosted on our cloud infrastructure. As a result, whenever a user makes a request to a Google service, the user agent generates a NEL report and attempts to upload it to one of our cloud collectors with a probability based on the values of the `weight` fields in our NEL policy. The user agent then generates and uploads a *meta report* about the upload of the original report.

Although this technique does not grant us visibility into problems affecting individual cloud tenants (*e.g.*, a misconfigured tenant firewall), it at least lets us detect problems affecting our entire cloud infrastructure, even if those problems are localized to a small number of clients. For example, this helped us quickly confirm that the BGP leak in Section 4.2 also impacted our cloud infrastructure.

One caveat to meta reports is that they are not representative. Any sampling rates defined in the NEL policy are compounded for meta reports, making it more difficult to get a large enough collection of meta reports to derive a statistically meaningful signal. Clients with unreliable connectivity are more likely to attempt to send NEL reports, and to fail

[6] https://news.ycombinator.com/item?id=13892431

doing so. So, we see higher baseline error rates for NEL report traffic from such clients compared to the global set of NEL reports. As a result, we cannot compare NEL error rates for our cloud infrastructure and our non-cloud infrastructure; but, trend analysis on meta reports remains useful.

5 Deployment Challenges

This section discusses several practical challenges we encountered when deploying NEL. Other web service operators are likely to encounter similar hurdles.

5.1 Collector diversity

As seen in some of our case studies (§4), diversity in the deployment of collectors has been crucial to collect client unreachability reports in a timely manner. For example, in the BGP leak case, it was crucial that we had a collector in a different prefix from those leaked.

While such diversity existed in Google's infrastructure even prior to our deployment of NEL, hosting collectors across the globe, in multiple prefixes and AS numbers, and supporting multiple IP versions is unlikely to be straightforward for an arbitrary web service. Therefore, to ease the use of NEL by other web services, we envision public cloud providers offering "NEL collectors as a service." Like Google, other large cloud providers also have a rich diversity of global infrastructure that naturally lends itself for use as NEL collectors.

5.2 Overhead

NEL increases network traffic for both clients and service providers in two ways: 1) the additional header that the service provider must include in its responses to clients' HTTP requests, and 2) reports that clients upload to NEL collectors.

Response header overhead. As shown in Figure 3, servers must send two headers to activate NEL: NEL and Report-To. Although exact sizes vary depending on the number of collectors and the presence of non-default options, headers are typically several hundred bytes long and are uncompressed unless the client is using HTTP/2. Furthermore, there is currently no way for clients to tell the server whether they support NEL or already have activated NEL for an origin, so servers typically include headers on all requests to all clients. This could be particularly problematic when serving many small objects, in which case NEL headers could constitute a significant fraction of the total response size.

Service operators have several ways to curtail NEL's bandwidth usage. Operators could (1) only serve NEL on a fixed fraction of responses, (2) try to predict which clients support NEL based on their User-Agent header, or (3) only serve NEL headers on HTTP/2 connections, under the assumption that (due to NEL's relatively young age) all clients that support NEL also support HTTP/2.

Report upload overhead. NEL also incurs overhead whenever clients upload reports. Reports we receive from our clients are 532 ± 34 bytes long; clients batch these into uploads that contain an average of 1.3 reports each. Clients upload a batch of reports about an origin once per minute.

Clients pay additional bandwidth overhead for failed uploads: 1) Clients incur connection establishment overhead multiple times as they retry an upload to different collectors, and 2) each failed upload may itself generate a NEL report.

Service operators can control these overheads with sampling rates in the NEL header. In particular, because *most* requests succeed, the success_fraction field can have a large impact on total upload traffic. For example, over 90% of requests to our services succeed and reports about those requests do not contribute much to our ability to reason about network outages other than by establishing proper baselines. We set success_fraction to 0.05, but success reports are still almost 40% of our upload traffic.

5.3 Provisioning for bursty workloads

We could further reduce success_fraction, but at a cost. Although request failures are much rarer than successes, failures are very bursty. Major network outages can cause large networks to send tens or hundreds of times more NEL reports than they normally would. A service's NEL collection infrastructure must be provisioned to handle these cases or risk data collection failing at exactly when it is most needed.

NEL client retry logic compounds this by causing clients to retry uploads to collectors when they are overloaded. We currently mitigate this by having our collectors always return HTTP 200, which prevents clients from retrying uploads when the collection infrastructure is overloaded. If more explicit control is needed, the NEL specification requires user agents to stop sending reports to a collector entirely when that collector returns an HTTP 410 (*Gone*) response.

5.4 Application-layer retries

It can be tempting to compare error rates across different kinds of applications, domains, and URLs. For example, one might suspect that if one domain has four times the error rate of another then something must surely be wrong with the former domain. Although this *might* be true, application-layer retry logic could also explain the discrepancy.

For example, if a Web application makes a lot of AJAX requests to example.com and *retries those requests when they fail*, then the overall NEL error rate for example.com will appear higher. This is because successive request failures are likely not independent; if a request fails once, it is much more likely that a second request will also fail.

This logic also applies to user-initiated retries. If a particular request is more likely to elicit a retry (*e.g.*, a browser reload) from a user when it fails, that can also inflate the NEL error rate. For example, a user may be more likely to retry requests that are very important to them whereas they may simply abandon more trivial requests.

6 Future Work

End-to-end report encryption. As mentioned in Section 5.1, one easy way operators can increase the likelihood that clients can successfully upload NEL reports is to host collectors in cloud providers. However, NEL clients currently assume that operators trust collectors and therefore do not encrypt reports beyond uploading them using HTTPS. If an operator does not trust a cloud, their only option currently is to use the cloud as a layer 3 proxy and terminate HTTPS elsewhere. Future versions of NEL could encrypt reports end-to-end (*e.g.*, using PGP), which would decouple collectors (which would see reports as opaque blobs) and report analyzers (which could decrypt reports).

Automated triage. Individual NEL reports are rarely useful; we derive utility from examining collections of reports and identifying patterns in those collections. For example, if we see many timeouts of requests to many IPs in a prefix, that may indicate a problem with that entire prefix. We currently look for problems in a manually curated set of dimensions based on our past experience, but in the future could try to automatically identify problematic user populations without *a priori* knowledge of likely problems. Based on the "shape" of a given problem (*i.e.*, the distribution of different types of NEL reports), we could attempt to automate triage of the problem. For example, if we detect that all users in just one ISP are having difficulty accessing a domain, then we could automatically notify that ISP, since their network configuration is a likely culprit.

Reducing overhead. Future versions of NEL might add several mechanisms to reduce the overhead of NEL policy headers. For example, we might let services specify NEL policies in external URLs, using a mechanism like the proposed Origin Policy [33] or Site-Wide HTTP Headers [26] standards. By making their NEL policy object cacheable, service providers can preclude clients from having to fetch the policy from the external URL after every successful request. We may also let clients include request headers which indicate when the server should send NEL policy headers, using a mechanism similar to HTTP Client Hints [16]. This would prevent servers from needlessly sending NEL headers to clients that will ignore them.

7 Related Work

Earlier in Section 2, we reviewed prior work which shares NEL's aim of detecting and diagnosing service unreachability. Here, we compare NEL to other systems which also rely on client-side measurements.

There have been a number of previous client-side measurement systems focused on one of the following goals: continual collection of passive measurements [30], enabling users to measure their network [21], or orchestration of measurement campaigns [29, 25, 13]. All of these efforts aim to gather measurements with the aim of compiling performance distributions, characterizing middleboxes, measuring network topologies, etc. Since none of this data needs to be compiled in a timely manner, uploading via redundant paths has been unnecessary in these efforts, unlike in NEL. Moreover, since the measurements gathered in these systems do not contain application traffic, protecting user privacy has not been a concern.

Windows Error Reporting (WER) [15] is most similar in spirit to NEL in that it too uploads error reports gathered at the client. WER does have to pay attention to privacy by pruning crash reports before uploading them. However, since uploaded crash reports are analyzed at a later point in time [18], failover on the upload path was unnecessary in this case too.

Odin [10] enables Microsoft to gather measurements from their clients to their CDN. By incorporating active measurement logic into client applications, Odin preempts concerns regarding coverage associated with dedicated monitoring infrastructure. Like NEL, Odin too attempts to make report uploading fault-tolerant via proxies in third-party networks. Unlike NEL, since Odin only relies on measurements that it actively issues, purging reports to protect privacy is not a significant concern. Odin also cannot be used by services not managed by Microsoft.

8 Conclusion

Despite the wide range of solutions available today to detect and diagnose reachability issues over the Internet, service operators lack an important capability: the ability to quantify the number of clients affected by any particular outage. To fill this void, we presented NEL, which equips service providers with timely collection of reachability reports generated at the client. Incorporation of NEL's techniques into Chrome has proved invaluable over the last few years in monitoring reachability to Google's domains. Motivated by our experience, we have proposed NEL as a W3C standard to spur support for it in other user agents and to enable other service providers to benefit from this capability.

Acknowledgements

We thank the anonymous reviewers and our shepherd, KyoungSoo Park, for their valuable feedback. Many people at Google have contributed to this project, particularly through their experiences debugging problems detected by NEL. In particular, we thank Emma Christie, Fred Douglas, Logan Ingalls, Martijn Stevenson, Matthew Steele, Steve Wang, Wesley Darlington, and Yan Avlasov for investigating issues that NEL has detected over the years. Debashish Chatterjee, Francesco Marabotto, Jelmer Vernooij, and Mitchell Jeffrey have provided valuable support from Google's SRE organization. We thank Chris Bentzel for initial sponsorship of the project.

References

[1] BGP errors are to blame for Monday's Twitter outage, not DDoS attacks. https://www.csoonline.com/article/3138934/security/bgp-errors-are-to-blame-for-monday-s-twitter-outage-not-ddos-attacks.html.

[2] A DNS hijacking wave is targeting companies at an almost unprecedented scale. https://arstechnica.com/information-technology/2019/01/a-dns-hijacking-wave-is-targeting-companies-at-an-almost-unprecedented-scale/.

[3] Dynatrace. http://www.dynatrace.com.

[4] ELK stack: Elasticsearch, Logstash, Kibana. https://www.elastic.co/elk-stack.

[5] Google Stackdriver. https://cloud.google.com/stackdriver/.

[6] RIPE Atlas. https://atlas.ripe.net.

[7] ThousandEyes. https://www.thousandeyes.com/solutions/bgp-and-route-monitoring.

[8] A. Barth. RFC 6454: The Web Origin Concept. *Internet Engineering Task Force (IETF)*, Dec. 2015.

[9] P. Anastas, W. R. Breen, Y. Cheng, A. Lieberman, and I. Mouline. Methods and apparatus for real user monitoring, 2010. US Patent 7,765,295.

[10] M. Calder, R. Gao, M. Schröder, R. Stewart, J. Padhye, R. Mahajan, G. Ananthanarayanan, and E. Katz-Bassett. Odin: Microsoft's scalable fault-tolerant CDN measurement system. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[11] D. Creager, I. Grigorik, A. Reitbauer, J. Tuttle, A. Jain, and J. Mann. Network Error Logging. W3C Editor's Draft, W3C Web Performance Working Group, 2019.

[12] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé. Analysis of country-wide internet outages caused by censorship. In *ACM Internet Measurement Conference (IMC)*, 2011.

[13] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A browser-based network measurement platform. In *ACM Internet Measurement Conference (IMC)*, 2012.

[14] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson. Examining how the great firewall discovers hidden circumvention servers. In *ACM Internet Measurement Conference (IMC)*, 2015.

[15] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt. Debugging in the (very) large: Ten years of implementation and experience. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[16] I. Grigorik and Y. Weiss. HTTP Client Hints. Internet-Draft, HTTP Working Group, 2019. https://httpwg.org/http-extensions/client-hints.html.

[17] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). Technical report, IETF, 2000.

[18] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie. Performance debugging in the large via mining millions of stack traces. In *International Conference on Software Engineering (ICSE)*, 2012.

[19] X. Hu and Z. M. Mao. Accurate real-time identification of IP prefix hijacking. In *IEEE Symposium on Security and Privacy*, 2007.

[20] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. E. Anderson. Studying black holes in the Internet with Hubble. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[21] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (IMC)*, 2010.

[22] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. PHAS: A prefix hijack alert system. In *USENIX Security symposium*, 2006.

[23] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, and J. Bailey. The QUIC transport protocol: Design and Internet-scale deployment. In *ACM SIGCOMM*, 2017.

[24] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *ACM SIGCOMM*, 2002.

[25] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *MobiSys*, 2015.

[26] M. Nottingham. Site-Wide HTTP Headers. Internet-Draft, IETF Network Working Group, 2017. https://mnot.github.io/I-D/site-wide-headers/.

[27] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[28] P. Richter, R. Padmanabhan, N. Spring, A. Berger, and D. Clark. Advancing the art of Internet edge outage detection. In *ACM Internet Measurement Conference (IMC)*, 2018.

[29] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing experiments to the Internets edge. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[30] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato. BISmark: A testbed for deploying measurements and applications in broadband access networks. In *USENIX ATC*, 2014.

[31] The Apache Software Foundation. Apache log files. https://httpd.apache.org/docs/2.4/logs.html.

[32] A. van Kesteren. Cross-origin resource sharing. Tech-

nical report, W3C, 2014.

[33] M. West. Origin Policy. Draft Community Group Report, Web Incubator Community Group, May 2017. https://wicg.github.io/origin-policy/.

[34] M. Zamani, J. Saia, and J. R. Crandall. TorBricks: Blocking-resistant Tor bridge distribution. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2017.

[35] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush. iSPY: Detecting IP prefix hijacking on my own. *IEEE/ACM Transactions on Networking*, 18(6):1815–1828, 2010.